HAL
open science

Simulating Induction-Recursion for Partial Algorithms

Dominique Larchey-Wendling, Jean-François Monin

To cite this version:

Dominique Larchey-Wendling, Jean-François Monin. Simulating Induction-Recursion for Partial Algorithms. 24th International Conference on Types for Proofs and Programs,TYPES 2018, Jun 2018, Braga, Portugal. hal-02333374

HAL Id: hal-02333374

https://hal.science/hal-02333374v1

Submitted on 25 Oct 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Simulating Induction-Recursion for Partial Algorithms

Dominique Larchey-Wendling[1] and Jean-François Monin[2]

[1] Université de Lorraine, CNRS, LORIA
dominique.larchey-wendling@loria.fr
[2] Université Grenoble Alpes, CNRS, Grenoble INP, VERIMAG
jean-francois.monin@univ-grenoble-alpes.fr

Abstract

We describe a generic method to implement and extract partial recursive algorithms in Coq in a purely constructive way, using L. Paulson's `if-then-else` *normalization as a running example.*

Implementing complicated recursive schemes in a Type Theory such as Coq is a challenging task. A landmark result is the Bove&Capretta approach [BC05] based on accessibility predicates, and in case of nested recursion, simultaneous Inductive-Recursive (IR) definitions of the domain/function [Dyb00]. Limitations to this approach are discussed in e.g. [Set06, BKS16]. We claim that the use of (1) IR, which is still absent from Coq, and (2) an informative predicate (of sort `Set` or `Type`) for the domain, preventing its erasing at extraction time, can be circumvented through a suitable *bar inductive predicate*.

$$
\begin{array}{l}
\texttt{type}\ \Omega = \alpha \mid \omega\ \texttt{of}\ \Omega * \Omega * \Omega \\
\texttt{let rec nm}\ e = \texttt{match}\ e\ \texttt{with} \\
\quad \mid \alpha \qquad\qquad\qquad\quad \Rightarrow \alpha \\
\quad \mid \omega(\alpha, y, z) \qquad\quad \Rightarrow \omega(\alpha, \texttt{nm}\ y, \texttt{nm}\ z) \\
\quad \mid \omega(\omega(a,b,c), y, z) \Rightarrow \texttt{nm}(\omega(a, \texttt{nm}(\omega(b,y,z)), \texttt{nm}(\omega(c,y,z))))
\end{array}
$$

Figure 1: L. Paulson's `if-then-else` normalisation algorithm.

We illustrate our technique on L. Paulson's algorithm for `if-then-else` normalization [Gie97, BC05] displayed in Fig. 1. For concise statements, we use ω to denote the ternary constructor for `if␣then␣else␣` expressions, and α as the nullary constructor for atoms. As witnessed in the third match rule $\omega(\omega(a,b,c),y,z)$, `nm` contains (two) nested recursive calls, making its termination depend on properties of its semantics. This circularity complicates the approach of well-founded recursion and may even render it unfeasible.

Our method allows to show these properties *after* the (partial) function `nm` is defined, as proposed in [Kra10], but without the use of Hilbert's ε-operator. We proceed purely constructively without any extension to the existing Coq system and the recursive definition of Fig. 1 can be *extracted as is* from the Coq term that implements `nm`.

We start with the inductive definition of the graph $\mathbb{G} : \Omega \to \Omega \to \texttt{Prop}$ of `nm` (Fig. 2) and we show its functionality.[1] Then we define the domain/termination predicate $\mathbb{D} : \Omega \to \texttt{Prop}$ as a bar inductive predicate with the three rules of Fig. 3.

$$
\frac{}{\mathbb{G}\ \alpha\ \alpha}
\qquad
\frac{\mathbb{G}\ y\ n_y \qquad \mathbb{G}\ z\ n_z}{\mathbb{G}\ (\omega\ \alpha\ y\ z)\ (\omega\ \alpha\ n_y\ n_z)}
$$

$$
\frac{\mathbb{G}\ (\omega\ b\ y\ z)\ n_b \qquad \mathbb{G}\ (\omega\ c\ y\ z)\ n_c \qquad \mathbb{G}\ (\omega\ a\ n_b\ n_c)\ n_a}{\mathbb{G}\ (\omega\ (\omega\ a\ b\ c)\ y\ z)\ n_a}
$$

Figure 2: Rules for the graph $\mathbb{G} : \Omega \to \Omega \to \texttt{Prop}$ of `nm`.

$$
\frac{}{\mathbb{D}\ \alpha}
\qquad
\frac{\mathbb{D}\ y \qquad \mathbb{D}\ z}{\mathbb{D}\ (\omega\ \alpha\ y\ z)}
$$

$$
\frac{\begin{array}{c}\mathbb{D}\ (\omega\ b\ y\ z) \qquad \mathbb{D}\ (\omega\ c\ y\ z) \\ \forall n_b\ n_c, \mathbb{G}\ (\omega\ b\ y\ z)\ n_b \to \mathbb{G}\ (\omega\ c\ y\ z)\ n_c \to \mathbb{D}\ (\omega\ a\ n_b\ n_c)\end{array}}{\mathbb{D}\ (\omega\ (\omega\ a\ b\ c)\ y\ z)}
$$

Figure 3: Rules for the bar inductive definition of $\mathbb{D} : \Omega \to \texttt{Prop}$.

There, we single out recursive calls using $\mathbb{G}$ but proceed by pattern-matching on e following the recursive scheme of `nm` of Fig. 1. Then we define $\texttt{nm_rec} : \forall e\ (D_e : \mathbb{D}\ e), \{n \mid \mathbb{G}\ e\ n\}$ as a fixpoint using D_e to ensure termination. However, the term $\texttt{nm_rec}\ e\ D_e$ does not use D_e to compute: the value n satisfying $\mathbb{G}\ e\ n$ is computed by pattern-matching on e and recursion, following the scheme of Fig. 1.

Finally, we define $\texttt{nm}\ e\ D_e := \pi_1(\texttt{nm_rec}\ e\ D_e)$ and get $\texttt{nm_spec}\ e\ D_e : \mathbb{G}\ e\ (\texttt{nm}\ e\ D_e)$ using the second projection π_2. Extraction of OCaml code from `nm` outputs exactly the algorithm of Fig. 1, illustrating the purely logical (`Prop`) nature of D_e. In order to reason on $\mathbb{D}/\texttt{nm}$ we show that they satisfy the IR specification given in Fig. 4: the constructors of $\mathbb{D}$ are sufficient to establish the simulated constructors `d_nm_[012]`, while `nm_spec` allows us to derive the fixpoint equations of `nm`. Us-

[1] i.e. $\texttt{g_nm_fun} : \forall e\ n_1\ n_2, \mathbb{G}\ e\ n_1 \to \mathbb{G}\ e\ n_2 \to n_1 = n_2$.

ing g_nm_fun, we get proof-irrelevance of nm.[2]

```
Inductive Ω : Set := α : Ω | ω : Ω → Ω → Ω → Ω.
Inductive 𝔻 : Ω → Prop :=
  | d_nm_0                  : 𝔻 α
  | d_nm_1 y z              : 𝔻 y → 𝔻 z → 𝔻(ω α y z)
  | d_nm_2 a b c y z D_b D_c : 𝔻(ω a (nm (ω b y z) D_b)
                                  (nm (ω c y z) D_c))
                            → 𝔻(ω (ω a b c) y z)
with Fixpoint nm e (D_e : 𝔻 e) : Ω := match D_e with
  | d_nm_0                  ↦ α
  | d_nm_1 y z D_y D_z      ↦ ω α (nm y D_y) (nm z D_z)
  | d_nm_2 a b c y z D_b D_c D_a ↦ nm (ω a (nm (ω b y z) D_b)
                                      (nm (ω c y z) D_c)) D_a
 end.
```

Figure 4: IR spec. of $\mathbb{D} : \Omega \to \texttt{Prop}$ and $\texttt{nm} : \forall e, \mathbb{D}\ e \to \Omega$.

We show a dependent induction principle for $\mathbb{D}$ (see Fig. 5). The term d_nm_ind states that any dependent property $P : \forall e, \mathbb{D}\ e \to \texttt{Prop}$ contains $\mathbb{D}$ as soon as it is closed under the simulated constructors d_nm_[012] of $\mathbb{D}$. The assumption $\forall e\, D_1\, D_2, P\ e\ D_1 \to P\ e\ D_2$ restricts the principle to *proof-irrelevant* properties about the dependent pair (e, D_e). This is exactly what we need to establish properties of nm. Then we can show partial correctness and termination as in [Gie97] – in this example, nm happens to always terminate on a normal form of its input. In a more relational approach, these properties can alternatively be proved using nm_spec and induction on $\mathbb{G}\ x\ n_x$.

```
Theorem d_nm_ind (P : ∀e, 𝔻 e → Prop) :
    (∀e D_1 D_2, P e D_1 → P e D_2) → (P _ d_nm_0)
  → (∀y z D_y D_z, P _ D_1 → P _ D_z → P _ (d_nm_1 y z D_y D_z))
  → (∀a b c y z D_b D_c D_a, P _ D_b → P _ D_c → P _ D_a ...
        ... → P _ (d_nm_2 a b c y z D_b D_c D_a))
  → ∀e D_e, P e D_e.
```

Figure 5: Dependent induction principle for $\mathbb{D} : \Omega \to \texttt{Prop}$.

Though our approach is inspired by IR definitions, in contrast with previous work, e.g. [Bov09], the corresponding principles are established *independently* of any consideration on the semantics or termination of the target function (nm), i.e. without proving any properties of $\mathbb{D}/\texttt{nm}$ a priori. This postpones the study of termination after both $\mathbb{D}$ and nm are defined together with constructors and elimination scheme, fixpoint equations and proof-irrelevance. Moreover, our domain/termination predicate $\mathbb{D}$ is *non-informative*, i.e. it does not carry any computational content. Thus the code obtained by extraction is exactly as intended.

[2] i.e. $\texttt{nm_pirr} : \forall e\, D_1\, D_2, \texttt{nm}\ e\ D_1 = \texttt{nm}\ e\ D_2$.

Our Coq code is available under a Free Software license [LWM18]. We have successfully implemented other algorithms using the same technique: F91, unification, depth first search as in [Kra10], quicksort, iterations until 0, partial list map as in [BKS16], Huet&Hullot's list reversal [Gie97], etc. The method is not constrained by nested/mutual induction, partiality or dependent types. On the other hand, spotting recursive sub-calls implies the explicit knowledge of all the algorithms that make such calls, a limitation that typically applies to higher order recursive schemes such as e.g. substitutions under binders. Besides growing our bestiary of examples, we aim at formally defining a class of schemes for which our method is applicable, and more practically propose some automation like what is done in Equations [Soz10].

References

[BC05] A. Bove and V. Capretta. Modelling general recursion in type theory. *Math. Struct. Comp. Science*, 15(4):671–708, 2005.

[BKS16] A. Bove, A. Krauss, and M. Sozeau. Partiality and recursion in interactive theorem provers - an overview. *Math. Struct. Comp. Science*, 26(1):38–88, 2016.

[Bov09] A. Bove. Another Look at Function Domains. *Electr. Notes Theor. Comput. Sci.*, 249:61–74, 2009.

[Dyb00] P. Dybjer. A General Formulation of Simultaneous Inductive-Recursive Definitions in Type Theory. *J. Symb. Log.*, 65(2):525–549, 2000.

[Gie97] J. Giesl. Termination of Nested and Mutually Recursive Algorithms. *J. Autom. Reasoning*, 19(1):1–29, 1997.

[Kra10] A. Krauss. Partial and Nested Recursive Function Definitions in Higher-order Logic. *J. Autom. Reasoning*, 44(4):303–336, 2010.

[LWM18] D. Larchey-Wendling and J.F. Monin. The If-Then-Else normalisation algorithm in Coq. https://github.com/DmxLarchey/ite-normalisation, 2018.

[Set06] A. Setzer. Partial Recursive Functions in Martin-Löf Type Theory. In *CiE 2006*, volume 3988 of *LNCS*, pages 505–515, 2006.

[Soz10] M. Sozeau. Equations: A Dependent Pattern-Matching Compiler. In *ITP 2010*, volume 6172 of *LNCS*, pages 419–434, 2010.